NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

Features (x)

Labels (y)

2 hidden layers

SOURCE
NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

\[ p = g_3(W_3g_2(W_2g_1(W_1x_0))) \]
NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

Table 2 – Results of both Models:

<table>
<thead>
<tr>
<th>Model not including past elections (Model A)</th>
<th>Model including past elections (Model B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D +3 )</td>
<td>( D +17 )</td>
</tr>
<tr>
<td>\textbf{Democrat Seats: 219}</td>
<td>\textbf{Democrat Seats: 226}</td>
</tr>
<tr>
<td>\textbf{Republican Seats: 216}</td>
<td>\textbf{Republican Seats: 209}</td>
</tr>
<tr>
<td>\textbf{33% chance of Republicans keeping house*}</td>
<td>\textbf{0.3% chance of Republicans keeping house*}</td>
</tr>
</tbody>
</table>

SOURCE
OK, SO NOW LET’S FIND THE WEIGHTS
OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]

\[ p = g_3(W_3g_2(W_2g_1(W_1x_0))) \]

NETWORK FUNCTION
OPTIMIZATION
[OR HOW TO FIND THE WEIGHTS?]

\[ p = g_3(W_3g_2(W_2g_1(W_1\vec{x}_0))) \]

\[ \frac{1}{N} \sum_{i=1}^{N} (y_i - p_i)^2 \] LOSS FUNCTION
WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT GENERATE THE MINIMUM LOSS
WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT GENERATE THE MINIMUM LOSS

WE THEN USE STANDARD MINIMIZATION ALGORITHMS THAT YOU ALL KNOW…
FOR EXAMPLE….

Gradient Descent

\[ W_{t+1} = W_t - \lambda_t \nabla f(W_t) \]

Newton

\[ W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t) \]

NEWTON CONVERGES FASTER…
FOR EXAMPLE….

Gradient Descent

\[ W_{t+1} = W_t - \lambda_t \nabla f(W_t) \]

[gradient]

Newton

\[ W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t) \]

[hessian]

NEWTON CONVERGES FASTER…

BUT NEEDS THE HESSIAN

\[
H = \begin{bmatrix}
\frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\
\frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2}
\end{bmatrix}
\]
FOR EXAMPLE...

Gradient Descent

\[ W_{t+1} = W_t - \lambda_t \nabla f(W_t) \]

[gradient]

Newton

\[ W_{t+1} = W_t - \lambda [Hf(W_t)]^{-1} \nabla f(W_t) \]

[hessian]

NEWTON CONVERGES FASTER...

BUT NEEDS THE HESSIAN

\[
H = \begin{bmatrix}
\frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\
\frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2}
\end{bmatrix}
\]
FOR EXAMPLE….

Gradient Descent

\[ W_{t+1} = W_t - \lambda_t \nabla f(W_t) \]

Newton

\[ W_{t+1} = W_t - \lambda [H f(W_t)]^{-1} \nabla f(W_t) \]

NEWTON CONVERGES FASTER…

BUT NEEDS THE HESSIAN

\[
H = \begin{bmatrix}
\frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1 \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1 \partial W_n} \\
\frac{\partial^2 f}{\partial W_2 \partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2 \partial W_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial W_n \partial W_1} & \frac{\partial^2 f}{\partial W_n \partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2}
\end{bmatrix}
\]
NICE, BUT I NEED TO COMPUTE THE GRADIENT AT EVERY ITERATION OF AN ARBITRARY COMPLEX FUNCTION!
BACKPROPAGATION

[AT THE NEURON LEVEL]

$\mathbf{x}$

activations

$\mathbf{y}$

$\mathbf{z}$

$\mathbf{f}$

Credit: A. Karpathy
BACKPROPAGATION

[AT THE NEURON LEVEL]

Credit: A. Karpathy
BACKPROPAGATION

[AT THE NEURON LEVEL]

Credit: A. Karpathy
BACKPROPAGATION
[AT THE NEURON LEVEL]

Credit: A. Karpathy
BACKPROPAGATION

[AT THE NEURON LEVEL]

Credit: A. Karpathy
BACKPROPAGATION
[AT THE NEURON LEVEL]

Credit: A. Karpathy
LET’S FOLLOW A NETWORK WHILE IT LEARNS…
LET’S ASSUME A VERY SIMPLE TRAINING SET:
\[ X=(0.05, 0.10) \rightarrow Y=(0.01,0.99) \]
1. THE FORWARD PASS

\[ i_{n_1} = w_1 i_1 + w_2 i_2 + b_1 \]

\[ i_{n_1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775 \]

[with initial weights]
1. THE FORWARD PASS

\[ i_{n h_1} = w_1 i_1 + w_2 i_2 + b_1 \]

\[ i_{n h_1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775 \]

[with initial weights]

\[ o_{u t h_1} = \frac{1}{1 + e^{-i_{n h_1}}} = 0.5932 \]

[after the activation function]
1. THE FORWARD PASS

WE CONTINUE TO $o_1$

$$in_{o_1} = w_5 out_{h_1} + w_6 out_{h_2} + b_2$$

$$in_{o_1} = 0.4 \times 0.593 + 0.45 \times 0.596 + 0.6 = 1.105$$

$$out_{o_1} = \frac{1}{1 + e^{-1.105}} = 0.751$$
1. THE FORWARD PASS

AND THE SAME FOR o2

\[ \text{out}_{o2} = 0.7729 \]
2. THE LOSS FUNCTION

\[ L_{total} = \sum 0.5(target - output)^2 \]

\[ L_{o1} = 0.5(target_{o1} - output_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274 \]

\[ L_{o2} = 0.023 \]
2. THE LOSS FUNCTION

\[ L_{total} = \sum 0.5(\text{target} - \text{output})^2 \]

\[ L_{o1} = 0.5(\text{target}_{o1} - \text{output}_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274 \]

\[ L_{o2} = 0.023 \]

\[ L_{total} = L_{o1} + L_{o2} = 0.298 \]
3. THE BACKWARD PASS

FOR W5 WE WANT: \[
\frac{\partial L_{total}}{\partial w_5}
\]
[gradient of loss function]
3. THE BACKWARD PASS

FOR $w_5$ WE WANT: \[ \frac{\partial L_{total}}{\partial w_5} \] [gradient of loss function]

WE APPLY THE CHAIN RULE:

\[ \frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5} \]
3. THE BACKWARD PASS

\[
\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}
\]

\[
L_{total} = 0.5(target_{o1} - out_{o1})^2 + 0.5(target_{o2} - out_{o2})^2
\]

\[
\frac{\partial L_{total}}{\partial out_{o1}} = 2 \times 0.5(target_{o1} - out_{o1}) \times (-1) = 0.741
\]
3. THE BACKWARD PASS

\[ \frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5} \]

\[ out_{o1} = \frac{1}{1 + e^{-in_{o1}}} \]

\[ \frac{\partial out_{o1}}{\partial in_{o1}} = out_{o1} \times (1 - out_{o1}) = 0.186 \]
3. THE BACKWARD PASS

\[
\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}
\]

\[
in_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2
\]

\[
\frac{\partial in_{o1}}{\partial w_5} = out_{h1} \times w_5^{1-1} = out_{h1} = 0.593
\]

\[
E_{o1} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2
\]

\[
E_{total} = E_{o1} + E_{o2}
\]
3. THE BACKWARD PASS

ALL TOGETHER:

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial \text{out}_{o1}} \times \frac{\partial \text{out}_{o1}}{\partial \text{in}_{o1}} \times \frac{\partial \text{in}_{o1}}{\partial w_5}$$

$$\frac{\partial L_{total}}{\partial w_5} = 0.741 \times 0.186 \times 0.593 = 0.082$$
4. UPDATE WEIGHTS WITH GRADIENT AND LEARNING RATE

\[ w_{5}^{t+1} = w_{5} - \lambda \times \frac{\partial L_{total}}{\partial w_{5}} \]

\[ w_{5}^{t+1} = 0.4 - 0.5 \times 0.082 = 0.358 \]
THIS IS REPEATED FOR THE OTHER WEIGHTS OF THE OUTPUT LAYER

\[ w_{6}^{t+1} = 0.408 \]

\[ w_{7}^{t+1} = 0.511 \]

\[ w_{8}^{t+1} = 0.561 \]
AND BACK-PROPAGATED TO THE HIDDEN LAYERS
VISUALIZE SIMPLE NETWORK LEARNING
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEEPST DESCENT
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

THE CHOICES OF THE INITIAL WEIGHTS AND THE LEARNING RATES ARE IMPORTANT
ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

THE CHOICES OF THE INITIAL WEIGHTS AND THE LEARNING RATES ARE IMPORTANT

WE WILL TALK ABOUT THIS LATER
LEARNING RATES

Credit:
LEARNING RATES

\[ W_{t+1} = W_t - \lambda \nabla f(W_t) \]

THERE ARE DIFFERENT WAYS TO UPDATE THE LEARNING RATE

Credit:
LEARNING RATES

\[ W_{t+1} = W_t - \lambda \nabla f(W_t) \]

THERE ARE DIFFERENT WAYS TO UPDATE THE LEARNING RATE

ADAGRAD:
THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

\[ W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t) \]

G IS A MATRIX CONTAINING ALL PREVIOUS GRADIENTS. WHEN THE GRADIENT BECOMES LARGE THE LEARNING RATE IS DECREASED AND VICE VERSA.

\[ G_{t+1} = G_t + (\nabla f)^2 \]

Credit:
LEARNING RATES

\[ W_{t+1} = W_t - \lambda \nabla f(W_t) \]

THERE ARE DIFFERENT WAYS TO UPDATE THE LEARNING RATE

RMSProp:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

\[ W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t) \]

SAME AS ADAGRAD BUT G IS CALCULATED BY EXPONENTIALLY DECAYING AVERAGE

\[ G_{t+1} = \lambda G_t + (1 - \lambda)(\nabla f)^2 \]

Credit:
ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

\[ G_{t+1} = \beta_2 G_t + (1 - \beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1 - \beta_1)(\nabla f) \]

\[ W_{t+1} = W_t - \frac{\lambda}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t \]

with:

\[ \hat{M}_{t+1} = \frac{M_t}{1 - \beta_1} \]

\[ \hat{G}_{t+1} = \frac{G_t}{1 - \beta_2} \]
ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

\[
G_{t+1} = \beta_2 G_t + (1 - \beta_2) (\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1 - \beta_1) (\nabla f)
\]

ONLY FOR YOUR RECORDS

\[
W_{t+1} = \frac{\hat{G}_{t+1}}{\sqrt{G_t + \epsilon}}
\]

with:

\[
\hat{M}_{t+1} = \frac{M_t}{1 - \beta_1} \quad \hat{G}_{t+1} = \frac{G_t}{1 - \beta_2}
\]
**IN KERAS:**

**RMSprop**

```python
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

**Arguments**

- **lr**: float >= 0. Learning rate.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

**References**

- **rmsprop**: Divide the gradient by a running average of its recent magnitude
IN KERAS:

**Adam**

```python
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

Adam optimizer.

Default parameters follow those provided in the original paper.

**Arguments**

- **lr**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

**References**

- Adam - A Method for Stochastic Optimization
- On the Convergence of Adam and Beyond
BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET
BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

MINI-BATCH GRADIENT DESCENT

\[ W_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)}) \]

THE GRADIENT IS COMPUTED OVER A BATCH OF SIZE B
THE EXTREME CASE IS TO COMPUTE THE GRADIENT ON EVERY TRAINING EXAMPLE.

\[ V_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)}) \]
Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.
CAN WE GO DEEP NOW?
CAN WE GO DEEP NOW?

ALMOST THERE…LET’S THINK FOR A MOMENT ABOUT WHAT WE PUT AS INPUT…
What do we put as input?

THIS IS WHAT MACHINES SEE
What do we put as input?

PRE-PROCESS DATA TO EXTRACT MEANINGFUL INFORMATION

THIS IS GENERALLY CALLED FEATURE EXTRACTION
Spiral!
Emission line!
Merger!
Clump!
AGN!
\[ f_{\mathbf{W}}(\mathbf{x}) = \mathbf{y} \]

**NETWORK FUNCTION**

\[ \text{sgn}[(u-v)-0.8*(v-j)-0.7] \]

**FEATURES**

\[(U-V, V-J)\]

**WEIGHTS**

\[ Q(0), SF(1) \]

**LABEL**

UVJ (U–V versus V–J)

(a) All

0.5 < z < 1.0

\[ \Delta A_V = 1.0 \]

log sSFR$_{UV, cor}$ [yr$^{-1}$]

-11.2 -10.6 -10.0 -9.4 -8.8 -8.2

Liu+18
Pre-Processing:
Dimension reduction

PCA or manual (colors, C, A, n …)

N parameters

Learning algorithm
(Neural Network, SVM…)

morphs. photoz’s
….
“CLASSICAL” MACHINE LEARNING

Pre-Processing:
Dimension reduction

N parameters

Learning algorithm
(Neural Network, SVM…)

morphs. photoz’s

PCA or manual (colors, C, A, n …)
In Astronomy

- Colors, Fluxes
- Shape indicators
- Line ratios, spectral features
- Stellar Masses, Velocity Dispersions

Requires specialized software before feeding the machine learning algorithm

IT IMPLIES A DIMENSIONALITY REDUCTION!
PHOTOMETRIC REDSHIFTS

SDSS

g r i z

Collister+08
EVERYTHING IS IN THE FEATURES....WHAT IF I IGNORED SOME IMPORTANT FEATURES?
EVERYTHING IS IN THE FEATURES....WHAT IF I IGNORED SOME IMPORTANT FEATURES?
Features 
(x)

Bad Weather, Known to Lower Turnout, Will Greet Many Voters
Rain can decrease voter numbers, which studies show tends to help Republicans. “I hope it rains hard tomorrow,” one Republican candidate said.

10h ago
Other general computer vision features [for images!]

- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

For many years computer vision researchers have been trying to find the most general features.
Other general computer vision features [for images!]

- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES THE BEST CLASSICAL SOLUTION [BEFORE 2012] WHERE BASED ON LOCAL FEATURES
HISTOGRAM OF ORIENTED GRADIENTS (HoG)

1. DIVIDE IMAGE INTO SMALL SPATIAL REGIONS CALLED CELLS

2. COMPUTE INTENSITY GRADIENTS OVER N DIRECTIONS [TYPICALLY 9 FOR IMAGE]

3. COMPUTE WEIGHTED 1-D HISTOGRAM OF ALL DIRECTIONS. A CELL IS REDUCED TO N NUMBERS
HISTOGRAM OF ORIENTED GRADIENTS (HoG)
HISTOGRAM OF ORIENTED GRADIENTS (HoG)

KEEP THIS IMAGE IN MIND FOR LATER…
What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING?

LET THE NETWORK FIND THE INFO
What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING?

LET THE NETWORK FIND THE INFO

LARGE DIMENSION SIGNALS SUCH AS IMAGES OR SPECTRA WOULD REQUIRE TREMENDOUSLY LARGE MODELS

A 512x512 image as input of a fully connected layer producing output of same size:

\[(512 \times 512)^2 = 7e10\]
FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOSES ALL INVARIANCE TO TRANSLATION AND IGNORES CORRELATION IN THE DATA
FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOSES ALL INVARIANCE TO TRANSLATION

SO?
LET THE MACHINE FIGURE THIS OUT ("unsupervised feature extraction")

LET’S GO A STEP FORWARD INTO LOOSING CONTROL....
PART III: CONVOLUTIONAL NEURAL NETWORKS
Discrete Convolution

1D:  
[Spectra]  
\[ f(x) \ast g(x) = \sum_{k=-\infty}^{k=+\infty} f(k) \cdot g(k - x) \]

2D:  
[Images]  
\[ f(x, y) \ast g(x, y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k, l) \cdot g(x - k, y - l) \]
DISCRETE CONVOLUTION

1D: [Spectra]
\[ f(x) \ast g(x) = \sum_{k=-\infty}^{k=+\infty} f(k) \cdot g(k - x) \]

2D: [Images]
\[ f(x, y) \ast g(x, y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k, l) \cdot g(x - k, y - l) \]

CONVOLUTION KERNEL
INPUT DATA
1-D CONVOLUTION

Input

\[ \begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array} \]

\[ x \]

Kernel

\[ \begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array} \]

\[ w \]
1-D CONVOLUTION

Input:

\[ 1 \quad 4 \quad -1 \quad 0 \quad 2 \quad -2 \quad 1 \quad 3 \quad 3 \quad 1 \]

Kernel:

\[ 1 \quad 2 \quad 0 \quad -1 \]

Output:

\[ 9 \]
1-D CONVOLUTION

Input

1 4 -1 0 2 -2 1 3 3 1

x

w

1 2 0 -1

9 0
1-D CONVOLUTION

Input:

\[
\begin{array}{ccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{x} & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array}
\]

\[
\begin{array}{cccc}
9 & 0 & 1 \\
\end{array}
\]
1-D CONVOLUTION

Input

\[
\begin{array}{cccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array}
\]

\[
\begin{array}{cccc}
9 & 0 & 1 & 3 \\
\end{array}
\]
1-D CONVOLUTION

Input

\[ 1 \quad 4 \quad -1 \quad 0 \quad 2 \quad -2 \quad 1 \quad 3 \quad 3 \quad 1 \]

\[ x \]

\[ 1 \quad 2 \quad 0 \quad -1 \]

\[ w \]

\[ 9 \quad 0 \quad 1 \quad 3 \quad -5 \]
1-D CONVOLUTION

Input

\[ \begin{array}{cccccccc}
 1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array} \]

\[ x \]

\[ w \]

\[ \begin{array}{cccc}
 1 & 2 & 0 & -1 \\
\end{array} \]

\[ \begin{array}{cccccc}
 9 & 0 & 1 & 3 & -5 & -3 \\
\end{array} \]
1-D CONVOLUTION

Input

1 4 -1 0 2 -2 1 3 3 1

x

W

9 0 1 3 -5 -3 6
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL

\[ w_0, w_1, w_2, w_3 \]

\[ f \left( \sum_i w_i x_i + b \right) \]
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL.
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL

WITH THE ADVANTAGE THAT THE SAME WEIGHTS ARE APPLIED TO ALL THE SIGNAL: TRANSLATION INVARIANCE
2-D CONVOLUTION

SAME IDEA, BUT THE KERNEL IS NOW 2D

KERNEL

INPUT (IMAGE)

OUTPUT

Credit: animations from https://github.com/vdumoulin/conv_arithmetic
2-D CONVOLUTION

SAME IDEA, BUT THE KERNEL IS NOW 2D

IN THE EXAMPLE: EACH 3x3 REGION GENERATES AN OUTPUT

\[ \text{Size}_{\text{output}} = \text{Size}_{\text{input}} - \text{Size}_{\text{kernel}} + 1 \]

Credit: animations from https://github.com/vdumoulin/conv_arithmetic
EQUIVALENT TO A NEURON WITH 9 INPUTS

WEIGHTS ARE CODED IN THE KERNEL
This is what the network learns!

Equivalent to a neuron with 9 inputs

Weights are coded in the kernel.
The key is again that the same weights are applied to all image regions.
The activation function at every kernel position is given by:

\[ z(x) = \text{relu}(w x + b) \]
IN DEEP NETWORKS ReLU is the most commonly used activation function - see Vanishing Gradient Problem

\[ z(x) = \text{relu}(wx + b) \]
CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

A COLOR IMAGE IS A TENSOR OF SIZE height x width x channels
CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

A COLOR IMAGE IS A TENSOR OF SIZE height x width x channels

Then the kernel has also 3 channels
IN ASTRONOMY …

IT OPENS THE DOOR TO ANALYZE MULTIPLE FILTERS () SIMULTANEOUSLY
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED
IN KERAS...

model = Sequential()

model.add(Convolution2D(4, 5, 5, activation="relu"))
Since convolutions output one scalar, they can be seen as an individual neuron with a receptive field limited to the kernel dimensions.
SINCE CONVOLUTIONS OUTPUT ONE SCALAR, THEY CAN BE SEEN AS AN INDIVIDUAL NEURON WITH A RECEPTIVE FIELD LIMITED TO THE KERNEL DIMENSIONS

THE SAME NEURON IS FIRED WITH DIFFERENT AREAS FROM THE INPUT
EXAMPLE OF 32 FILTERS LEARNED IN A CONVOLUTIONAL LAYER

(a) red channel  (b) green channel  (c) blue channel

Dieleman+16
EXAMPLE OF 32 FILTERS LEARNED IN A CONVOLUTIONAL LAYER

These are called feature maps.

(a) red channel

(b) green channel

(c) blue channel

Dieleman+16

These are called feature maps.
### Estimating Shapes and Number of Parameters

<table>
<thead>
<tr>
<th>Kernel Shape:</th>
<th>Padding:</th>
<th>Strides:</th>
</tr>
</thead>
<tbody>
<tr>
<td>((F, F, C_i^i, C_o^o))</td>
<td>(P)</td>
<td>(S)</td>
</tr>
</tbody>
</table>

**Output Size:**

\[
W_0 = (W^i - F + 2P)/S + 1
\]
OPTIONS: STRIDES

NO STRIDES

STRIDES
OPTIONS: PADDING

NO STRIDES

PADDING
ESTIMATING SHAPES AND NUMBER OF PARAMETERS

<table>
<thead>
<tr>
<th>KERNEL SHAPE:</th>
<th>PADDING:</th>
<th>STRIDES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>((F, F, C^i, C^o))</td>
<td>(P)</td>
<td>(S)</td>
</tr>
</tbody>
</table>

**OUTPUT SIZE:** \(W_0 = (W_i - F + 2P)/S + 1\)

**NUMBER OF PARAMETERS:** \((F \times F \times C^i + 1) \times C^o\)
ESTIMATING SHAPES AND NUMBER OF PARAMETERS

<table>
<thead>
<tr>
<th>KERNEL SHAPE:</th>
<th>PADDING:</th>
<th>STRIDES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>((F, F, C^i, C^o))</td>
<td>(P)</td>
<td>(S)</td>
</tr>
</tbody>
</table>

OUTPUT SIZE: \(W_0 = (W^i - F + 2P)/S + 1\)

NUMBER OF PARAMETERS: \((F \times F \times C^i + 1) \times C^o\)

---

the number of parameters increases fast!

32 filters of 5*5 on a color image —> 2432 parameters to learn
DOWNSAMLING

DOWNSAMLING IS APPLIED TO REDUCE THE OVERALL SIZE OF TENSORS
POOLING

CONVOLUTIONS ARE OFTEN FOLLOWED BY AN OPERATION OF DOWNSAMPLING [POOLING]

VERY SIMPLE OPERATION - ONLY ONE OUT OF EVERY N PIXELS ARE KEPT

OFTEN MATCHED WITH AN INCREASE OF THE FEATURE CHANNELS
TYPES OF POOLING

SUM POOLING
\[ y = \sum x_{uv} \]

SQUARE SUM POOLING
\[ y = \sqrt{\sum x_{uv}^2} \]

MAX POOLING
\[ y = \max(x_{uv}) \]
TYPES OF POOLING

SUM POOLING

\[ y = \sum x_{uv} \]

SQUARE SUM POOLING

\[ y = \sqrt{\sum x_{uv}^2} \]

MAX POOLING

\[ y = \max(x_{uv}) \]
MAX POOLING 1D

Input:

1 4 -1 0 2 -2 1 3 3 1

Output:

4

Credit: F. Fleuret
MAX POOLING 1D

Input

Output

Credit: F. Fleuret
MAX POOLING 1D

Credit: F. Fleuret
MAX POOLING 1D

Credit: F. Fleuret
MAX POOLING 1D

Input
1 4 -1 0 2 -2 1 3 3 1

Output
4 0 2 3 3

Credit: F. Fleuret
CONVNET OR CNN

A CONCATENATION OF MULTIPLE CONVOLUTIONAL BLOCKS
CONVNET OR CNN

L1 \rightarrow L2 \rightarrow L3 \rightarrow L4

EACH BLOCK TYPICALLY MADE OF:

- CONV
- ACTIVATION
- POOLING

(+dropout for training)
EXAMPLE OF VERY SIMPLE CNN

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

3 convolutional layers

Input Images (RGB)

conv (ReLu) + dropout (0.5)
W = 3488

conv (ReLu) + MaxPooling + dropout (0.25)
W = 51264

conv (ReLu) + MaxPooling + dropout (0.25)
W = 32896

conv (ReLu) + dropout (0.25)
W = 147584

Fully connected
W = 2367552

Domínguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

3 convolutional layers

KERNEL SIZE

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

3 convolutional layers

Input Images (RGB)

conv (ReLU) + dropout (0.5)
W = 3488

conv (ReLU) + MaxPooling + dropout (0.25)
W = 51264

conv (ReLU) + MaxPooling + dropout (0.25)
W = 32896

conv (ReLU) + dropout (0.25)
W = 147584

Fully connected
W = 2367552

DEPTH

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

3 convolutional layers

ReLu activation

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

3 convolutional layers

Input Images (RGB)

conv (ReLU) + dropout (0.5)  
W = 3488

conv (ReLU) + MaxPooling + dropout (0.25)  
W = 51264

conv (ReLU) + MaxPooling + dropout (0.25)  
W = 32896

conv (ReLU) + dropout (0.25)  
W = 147584

Fully connected
W = 2367552

Pooling

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

OVERALL:
- decrease of tensor size
- increase of depth

Dominguez-Sanchez+18
# Model definition

## Convolutional Layers

```python
# Initialize a Sequential model
model = Sequential()

# Add a Conv2D layer with 32 filters, kernel size 6x6, and 'same' padding
model.add(Convolution2D(32, 6, 6, border_mode='same',
                        input_shape=(img_channels, img_rows, img_cols)))

# Add an Activation layer with 'relu'
model.add(Activation('relu'))

# Add a Dropout layer with 50% dropout rate
model.add(Dropout(0.5))

# Add another Conv2D layer with 64 filters, kernel size 5x5, and 'same' padding
model.add(Convolution2D(64, 5, 5, border_mode='same'))

# Add another Activation layer with 'relu'
model.add(Activation('relu'))

# Add a MaxPooling2D layer with pool size 2x2
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add another Dropout layer with 25% dropout rate
model.add(Dropout(0.25))

# Add another Conv2D layer with 128 filters, kernel size 2x2, and 'same' padding
model.add(Convolution2D(128, 2, 2, border_mode='same'))

# Add another Activation layer with 'relu'
model.add(Activation('relu'))

# Add another MaxPooling2D layer with pool size 2x2
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add another Dropout layer with 25% dropout rate
model.add(Dropout(0.25))

# Add another Conv2D layer with 128 filters, kernel size 3x3, and 'same' padding
model.add(Convolution2D(128, 3, 3, border_mode='same'))

# Add another Activation layer with 'relu'
model.add(Activation('relu'))

# Add another Dropout layer with 25% dropout rate
model.add(Dropout(0.25))

# Fully Connected start here

# Flatten the output of the Conv2D layers
model.add(Flatten())

# Add a Dense layer with 64 units, 'relu' activation
model.add(Dense(64, activation='relu'))

# Add another Dropout layer with 50% dropout rate
model.add(Dropout(0.5))

# Add a Dense layer with 1 output unit, 'uniform' weight initializer, 'sigmoid' activation
model.add(Dense(1, init='uniform', activation='sigmoid'))

# Compile the model with 'binary_crossentropy' loss, 'adam' optimizer, and 'accuracy' metric
print("Compilation...")
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```
MERGERS

Q6: Merger
Accuracy = 97.1%
N_{train} = 5000
N_{test} = 903
N_{pos} = 103

BARS

Q3: Bar
Accuracy = 96.6%
N_{train} = 10000
N_{test} = 1618
N_{pos} = 137
EXAMPLE OF VERY SIMPLE CNN

OVERALL:
- decrease of tensor size
- increase of depth

Number of parameters

Dominguez-Sanchez+18
EXAMPLE OF VERY SIMPLE CNN

OVERALL:
- decrease of tensor size
- increase of depth

2 million of parameters for this very simple network!
### Checking the Number of Parameters / Layers with Keras

```python
model.summary()
```

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_1 (InputLayer)</td>
<td>(None, 16, 112, 112)</td>
<td>0</td>
</tr>
<tr>
<td>conv3d_1 (Conv3D)</td>
<td>(None, 16, 112, 112)</td>
<td>448</td>
</tr>
<tr>
<td>batch_normalization_1 (Batch)</td>
<td>(None, 16, 112, 112)</td>
<td>448</td>
</tr>
<tr>
<td>activation_1 (Activation)</td>
<td>(None, 16, 112, 112)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling3d_1 (MaxPooling3D)</td>
<td>(None, 16, 8, 56, 56)</td>
<td>0</td>
</tr>
<tr>
<td>conv3d_2 (Conv3D)</td>
<td>(None, 32, 8, 56, 56)</td>
<td>13856</td>
</tr>
<tr>
<td>batch_normalization_2 (Batch)</td>
<td>(None, 32, 8, 56, 56)</td>
<td>224</td>
</tr>
<tr>
<td>activation_2 (Activation)</td>
<td>(None, 32, 8, 56, 56)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling3d_2 (MaxPooling3D)</td>
<td>(None, 32, 4, 28, 28)</td>
<td>0</td>
</tr>
<tr>
<td>conv3d_3 (Conv3D)</td>
<td>(None, 64, 4, 28, 28)</td>
<td>55360</td>
</tr>
<tr>
<td>batch_normalization_3 (Batch)</td>
<td>(None, 64, 4, 28, 28)</td>
<td>112</td>
</tr>
<tr>
<td>activation_3 (Activation)</td>
<td>(None, 64, 4, 28, 28)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling3d_3 (MaxPooling3D)</td>
<td>(None, 64, 2, 14, 14)</td>
<td>0</td>
</tr>
<tr>
<td>activation_12 (Activation)</td>
<td>(None, 64, 2, 14, 14)</td>
<td>0</td>
</tr>
</tbody>
</table>

Total params: 70,448  
Trainable params: 70,056  
Non-trainable params: 392
IN THE REAL LIFE...

RESNET
IN THE REAL LIFE...

RESNET

DO WE NEED TO GO THIS DEEP FOR ASTRONOMY APPLICATIONS?

[34 layers - authors explored up to 1202!]

He+15
DEEPER TENDS TO BE BETTER…

ImageNet experiments

ILSVRC'15 ResNet: 3.57%
ILSVRC'14 GoogleNet: 6.7%
ILSVRC'14 VGG: 7.3%
ILSVRC'13: 11.7% (8 layers)
ILSVRC'12 AlexNet: 16.4% (8 layers)
Shallow: 25.8%
ILSVRC'11: 28.2%
THE PROBLEMS OF GOING “TOO DEEP”

• DEEP NETWORKS ARE MORE DIFFICULT TO OPTIMIZE

• NEED MORE DATA - MORE SUBJECT TO OVERFITTING

• AND ALSO NEED MORE TIME …
OVER-FITTING

THE TRAINING LOSS DECREASES

THE TEST STAYS CONSTANT OR INCREASES
DROPOUT
[Hinton+12]
- THE IDEA IS TO REMOVE NEURONS RANDOMLY DURING THE TRAINING
- ALL NEURONS ARE PUT BACK DURING THE TEST PHASE

(a) Standard Neural Net
(b) After applying dropout.
DROPOUT

WHY DOES IT WORK?

1. SINCE NEURONS ARE REMOVED RANDOMLY, IT AVOIDS CO-ADAPTATION AMONG THEMSELVES

2. DIFFERENT SETS OF NEURONS WHICH ARE SWITCHED OFF, REPRESENT A DIFFERENT ARCHITECTURE AND ALL THESE DIFFERENT ARCHITECTURES ARE TRAINED IN PARALLEL. FOR N NEURONS ATTACHED TO DROPOUT, THE NUMBER OF SUBSET ARCHITECTURES FORMED IS 2^N. SO IT AMOUNTS TO PREDICTION BEING AVERAGED OVER THESE ENSEMBLES OF MODELS.
DROPOUT

WITH A LITTLE BIT OF DROPOUT

Huertas-Company+15
CAPTURING THE MODEL UNCERTAINTY

NEURAL NETWORKS AS BAYESIAN MODELS

Denker&LEcun91, Neal+95, Graves+11, Kingma+15, Gal+15...

BNNs ADD A PRIOR DISTRIBUTION TO EACH WEIGHT - HARD TO TRAIN

GAL+15 SHOW THAT DROPOUT CAN BE USED TO ESTIMATE UNCERTAINTY
IMPLEMENTATION IN KERAS / TENSORFLOW

# Model definition

# Convolutional Layers

```python
model = Sequential()
model.add(Convolution2D(32, 6, 6, border_mode='same',
                        input_shape=(img_channels, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(64, 5, 5, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Convolution2D(128, 2, 2, border_mode='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(Activation('relu'))
model.add(Dropout(0.25))
```

# Fully Connected start here

```python
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(.5))
model.add(Dense(1, init='uniform', activation='sigmoid'))
print("Compilation...")
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```
VANISHING / EXPLODING GRADIENT PROBLEM

REMEMBER THAT:

\[ y_{i+1} = \sigma \left[ \sum w_i y_i \right] \]

output layer \( i+1 \)  
activation function  
weights  
output layer \( i \)
VANISHING / EXPLODING GRADIENT PROBLEM

WITH MANY LAYERS:

\[ y_n = \sigma \left( \ldots \sigma \left( \ldots \sigma \left( \sum w_0 x \right) \right) \right) \]
VANISHING/EXPLODING GRADIENT PROBLEM

\[ y_n = \sigma \left( \ldots \sigma \left( \ldots \sigma \left( \sum w_0 x \right) \right) \right) \]
VANISHING/EXPLODING GRADIENT PROBLEM

TRAINING BECOMES UNSTABLE
VERY SLOW OR NO CONVERGENCE

$x$

L1

$w_1$

L2

$w_2$

L3

$w_3$

... 

Ln

!!!
VANISHING/EXPLODING GRADIENT PROBLEM

IF WE ASSUME AN IDENTITY ACTIVATION FUNCTION:

\[ \hat{y} = x \prod_{n} w_i \]

with:

\[ w_i = \begin{pmatrix} w_{i}^0 & 0 \\ 0 & w_{i}^1 \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \]
VANISHING/EXPLODING GRADIENT PROBLEM

\[
\hat{y} = x \prod_{n} w_{i}
\]

\[
w_{i} = \begin{pmatrix} w_{i}^{0} & 0 \\ 0 & w_{i}^{1} \end{pmatrix}
\]

\[
x = \begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix}
\]

IF WEIGHTS ARE ALL INITIALIZED TO VALUES $\ll 1$:

\[
w_{i}^{L} \to 0
\]

VANISHING GRADIENT
VANISHING/EXPLODING GRADIENT PROBLEM

\[ w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix} \]

\[ \hat{y} = x \prod_{n} w_i \]

\[ x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \]

IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

\[ w_i^L \rightarrow \infty \]
VANISHING/EXPLODING GRADIENT PROBLEM

TRAINING BECOMES UNSTABLE
VERY SLOW OR NO CONVERGENCE

![Graphs showing training and test error over iterations for 20-layer and 56-layer models.](image-url)
WEIGHT INITIALIZATION IS A KEY POINT...

\[ z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n \]
WEIGHT INITIALIZATION IS A KEY POINT...

\[ z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n \]

THE LARGER \( n \), THE SMALLER WEIGHTS SHOULD BE...
WEIGHT INITIALIZATION IS A KEY POINT...

\[ z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n \]

THE LARGER \( n \), THE SMALLER WEIGHTS SHOULD BE...

ONE SIMPLE SOLUTION:

\[ \sigma^2(w_i) = \frac{1}{n} \]

number of inputs
WEIGHT INITIALIZATION IS A KEY POINT...

\[ z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n \]

THE LARGER \( n \), THE SMALLER the weights should be...

ONE SIMPLE SOLUTION: \[ \sigma^2(w_i) = \frac{1}{n} \]

variance

number of inputs
WEIGHT INITIALIZATION IS A KEY POINT...

For ReLU activation functions we typically use:

$$\sigma^2(w_i) = \frac{2}{n}$$

[He initialization, He+15]
WEIGHT INITIALIZATION IS A KEY POINT…

IMPLEMENTATION IN KERAS:

```python
initialization = 'he_normal'
act = 'relu'

model = Sequential()
model.add(Convolution2D(depth, conv_size, conv_size, activation=act, border_mode='same',
                        name = "conv%i"%(layer_n), init=initialization, W_constraint=constraint))
```
WEIGHT INITIALIZATION IS A KEY POINT...

IMPLEMENTATION IN KERAS:

```python
ing_initialization = 'he_normal'
act = 'relu'

model = Sequential()
model.add(Convolution2D(depth, conv_size, conv_size, activation=act, border_mode='same',
                        name = "conv%i"%(layer_n), init=initialization, W_constraint=constraint))
```

MANY OTHER INITIALIZATIONS AVAILABLE:

keras.initializers | https://keras.io/initializers/
BATCH NORMALIZATION
[SZEGEDY+15]

ANOTHER SOLUTION TO KEEP REASONABLE VALUES OF THE ACTIVATIONS IN DEEP NETWORKS

**BATCH NORMALIZATION** PREVENTS LOW OR LARGE VALUES BY RE-NORMALIZING THE VALUES BEFORE ACTIVATION FOR EVERY BATCH

\[
\hat{y}_i = \gamma \frac{y_i - E(y_i)}{\sigma(y_i)} + \beta
\]

INPUT

NORMALIZED INPUT

SCATTER
BATCH NORMALIZATION [SZEGEDY+15]

Batch normalization speeds up and stabilizes training.

As for the dropout, there is a different behavior between training and testing.

\[ \hat{y}_i = \gamma \frac{y_i - E(y_i)}{\sigma(y_i)} + \beta \]

- **Input**
- **Normalized Input**
- **Scatter**
BATCH NORMALIZATION
[SZEGERDY+15]

IN KERAS, IT IS IMPLEMENTED AS AN ADDITIONAL LAYER

Batch normalization layer (Ioffe and Szegedy, 2014).

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Arguments

```python
keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer)
```
THIS IS A CHANGE OF PARADIGM!
Learning algorithm

DATA

FEATURE LEARNING LAYERS

Raw data

prediction
FEATURE LEARNING LAYERS

THE LEARNING ALGORITHM CAN BE CHANGED

ANN

prediction

Raw data

DATA

DIMENSION REDUCTION

PCA or manual (colors, C, A, n ...)

THE LEARNING ALGORITHM CAN BE CHANGED

Raw data

DATA

FEATURE LEARNING LAYERS

ANN

prediction
DATA

FEATURE LEARNING LAYERS

SVM?

prediction

THE LEARNING ALGORITHM CAN BE CHANGED

Raw data
THE LEARNING ALGORITHM CAN BE CHANGED

Raw data → DATA

FEATURE LEARNING LAYERS

SVM? OR ANY OTHER LEARNING ALGORITHM → prediction
THE FEATURES CAN BE MANIPULATED OR COMBINED

Raw data

DATA

FEATURE LEARNING LAYERS

manual (colors, C, A, n, ...)

SVM? OR ANY OTHER LEARNING ALGORITHM

prediction
THE FEATURES CAN BE MANIPULATED OR COMBINED

Features Learned from another CNN...

FEATURE LEARNING LAYERS

DATA

manual (colors, C, A, n ...)

SVM? OR ANY OTHER LEARNING ALGORITHM

prediction
THIS IS A CHANGE OF PARADIGM!
ALSO FOR GALAXY MORPHOLOGY

[HUERTAS-COMPANY+14]

SVMs

[HUERTAS-COMPANY+15b]

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]

ALSO FOR GALAXY MORPHOLOGY

SVMs

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]

ALSO FOR GALAXY MORPHOLOGY

SVMs

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]

ALSO FOR GALAXY MORPHOLOGY

SVMs

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]

ALSO FOR GALAXY MORPHOLOGY

SVMs

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]

ALSO FOR GALAXY MORPHOLOGY

SVMs

CNNs

AUTOMATIC

Late-Type

Early-Type

VISUAL

AUTOMATIC

PS

SPHEROID

DISK

IRR

Unc

SVMs

CNNs

Late-Type

Early-Type

99

87

13

25

75

25

87

25

13

[HUERTAS-COMPANY+15b]
PHOTOMETRIC REDSHIFTS

Pasquet+18

AUTOMATICALLY COMBINING MORPHOLOGY AND COLOR FOR PHOTOZ ESTIMATION
DATA QUALITY SELECTION FOR EUCLID

Thanks to H. McCracken
DATA QUALITY SELECTION FOR EUCLID

```python
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                          input_shape=(150, 150, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Thanks to H. McCracken
WELL, BUT THIS IS AN “OLD” IDEA - WHY NOW?
WELL, BUT THIS IS AN “OLD” IDEA - WHY NOW?

1 - MORE DATA TO TRAIN! DEEP NETWORKS HAVE A LARGE NUMBER OF PARAMETERS - THX TO SOCIAL MEDIA …
WELL, BUT THIS IS AN “OLD” IDEA - WHY NOW?

2 - GPUs - TRAINING OF THESE DEEP NETWORKS HAS REMAINED PROHIBITIVELY TIME CONSUMING WITH CPUs - THX TO VIDEO GAMES…
GPUs

NVIDIA TITANX GPU
GPUs vs. CPUs

CPUs
FEWER CORES (~10x)
EACH CORE IS FASTER
USEFUL FOR SEQUENTIAL TASKS

GPUs
MORE CORES (100x)
EACH CORE IS SLOWER
USEFUL FOR PARALLEL TASKS
GPUs vs. CPUs

More benchmarks available here.

Figure credit: J. Johnson
GPUs for deep learning

NVIDIA GPUs ARE PROGRAMMED THROUGH CUDA
[Compute Unified Device Architecture]

ANOTHER ALTERNATIVE IS OPENCL, SUPPORTED BY SEVERAL MANUFACTURES, LESS INVESTMENT [Way less used]

CuDNN IS A LIBRARY FOR SPECIFIC DEEP LEARNING COMPUTATIONS ON NVIDIA GPUs
THE PRICE TO PAY?

1. LARGE NUMBER OF PARAMETERS IMPLIES LARGE DATASETS TO TRAIN

2. LOOSE EVEN MORE DEGREE OF CONTROL OF WHAT THE ALGORITHM IS DOING SINCE THE FEATURE EXTRACTION PROCESS BECOMES UNSUPERVISED
IMAGE OF THE BACK OF THE EYE
IMAGE OF THE BACK OF THE EYE

DEEP LEARNING CAN IDENTIFY THE PATIENT’S GENDER WITH 95% ACCURACY
VISUALIZING CNNs
[what happens inside a CNN?]
DEEP NETWORKS ARE “BLACK BOXES”?

INTERPRETING THE RESULTS IS EXTREMELY DIFFICULT

THIS IS TRUE BUT A LOT OF WORK IS DONE TO UNVEIL THEIR BEHAVIOR
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

NOT VERY INFORMATIVE THOUGH...
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

```python
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
        input_shape=(img_channels, img_rows, img_cols),
        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

give names to layers

```python
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act, border_mode='same', name='conv0',
                        input_shape=(img_channels, img_rows, img_cols),
                        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

```python
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
            input_shape=(img_channels, img_rows, img_cols),
            init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])
layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```
THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

```python
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
    input_shape=(img_channels, img_rows, img_cols),
    init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```

for a given name, get the weights
USING THE SAME IDEA, ONE CAN ALSO VISUALIZE THE FEATURE MAPS AT INTERMEDIATE LAYERS

THIS HELPS TRACING THE FEATURES LEARNED BY THE NETWORK
USE “DECONVNETS” TO MAP BACK THE FEATURE MAP INTO THE PIXEL SPACE

IT ALLOWS TO SEE WHICH REGIONS OF THE INPUT GENERATED A MAXIMUM RESPONSE IN A NEURON

Zeiler+14
EVERY BLOCK OF 9 SHOWS
THE 9 STRONGEST RESPONSES TO A GIVEN FILTER OF LAYER 2
THE CORRESPONDING REGIONS OF IMAGES THAT GENERATED THE MAXIMUM RESPONSE
CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS INTUITIVE

Zeiler+14
CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS

Zeiler+14
KERAS IMPLEMENTATION OF VISUALIZATIONS THROUGH DECONVNETS

https://github.com/jalused/Deconvnet-keras
OCCLUSION SENSITIVITY TRIES ALSO TO FIND THE REGION OF THE IMAGE THAT TRIGGERED THE NETWORK DECISION BY MASKING DIFFERENT REGIONS OF THE INPUT IMAGE AND ANALYZING THE NETWORK OUTPUT.

IT ALLOWS TO IF THE NETWORK IS TAKING THE DECISIONS BASED ON THE EXPECTED FEATURES.

VERY TIME CONSUMING!

Zeiler+14
OCCLUSION SENSITIVITY TRIES ALSO TO FIND THE REGION OF THE IMAGE THAT TRIGGERED THE NETWORK DECISION BY MASKING DIFFERENT REGIONS OF THE INPUT IMAGE AND ANALYZING THE NETWORK OUTPUT.

For every position of the square, the maximum response of a given layer is averaged.

The output probability as a function of the occluding square position.

Zeiler+14
THE IDEA BEHIND INCEPTIONISM TECHNIQUES IS TO INVERT THE NETWORK TO GENERATE AN IMAGE THAT MAXIMIZES THE OUTPUT SCORE

\[
\arg \max_i S_c(I) - \lambda \|I\|^2_2
\]

TRY TO FIND AN IMAGE THAT GENERATES A HIGH SCORE FOR A GIVEN CLASS

Simonyan+14
DURING THE TRAINING PHASE THE WEIGHTS ARE LEARNED TO MAP $I$ INTO $S_c$
DURING THE RECONSTRUCTION PHASE, I IS LEARNT THROUGH BACKPROPAGATION KEEPING THE WEIGHTS FIXED
INCEPTIONISM - DEEP DREAM

RESULTS REVEAL INTERESTING INFORMATION ON HOW THE NETWORKS BUILD REPRESENTATIONS OF OBJECTS
INCEPTIONISM - DEEP DREAM

RESULTS REVEAL INTERESTING INFORMATION ON HOW THE NETWORKS BUILD REPRESENTATIONS OF OBJECTS

SOME STRANGE CASES…
DEEP DREAM

https://deepdreamgenerator.com/

IT HAS NOW BECOME A SORT OF ART?
INTEGRATED GRADIENTS

Integrated Gradient Visualization

Original Image  Perturbed Image  Sensitivity Map
INTEGRATED GRADIENTS

Pre-Compaction  
Compaction  
Post-Compaction
INTEGRATED GRADIENTS

KERAS IMPLEMENTATION:
https://github.com/hiranumn/IntegratedGradients
PART IV: IMAGE 2 IMAGE NETWORKS + INTRODUCTION TO GENERATIVE MODELS